Tweet Sentiment Classification

Mladen Korunoski **Ljupche Milosheski** **Konstantinos Ragios**
École Polytechnique Fédérale de Lausanne
{mladen.korunoski, ljupche.milosheski, konstantinos.ragios}@epfl.ch

Abstract—**We present an extensive study on binary text classification using data from the social networking service Twitter. The task was to predict whether a preprocessed tweet message used to contain positive or negative smiley, by only considering the remaining text. We developed an end-to-end text classification pipeline that contains data preparation, feature engineering, model training, and performance improvements using classical machine learning as well as deep learning architectures for text classification. We experimented with different approaches using the aforementioned techniques and obtained a classification accuracy of 88.3% with an ensemble of deep learning models, which outperforms the baseline Logistic Regression model by 6.54%.**

I. Introduction

Text classification is the process of assigning tags or categories to text according to its content. It is a fundamental task in natural language processing (NLP) with broad applications such as sentiment analysis, topic labelling, spam detection, and intent detection.

Unstructured data in the form of text is everywhere: emails, chats, web pages, social media, support tickets, survey responses, and more. Text can be an extremely rich source of information, but extracting insights from it can be hard and time-consuming due to its unstructured nature. Businesses are turning to text classification for structuring text in a fast and cost-efficient way to enhance decision-making and automate processes. Although the idea is simple, it is not a trivial problem and a lot of tech companies and research facilities work towards developing novel methods for accurate text classification. The problem is even more difficult when the text corpora is unconventional, such as the one from Twitter, with words, phrases, and abbreviations invented on the fly. Fortunately, with the advancements in machine learning, automatic learning procedures that make use of statistical inference algorithms produce models robust to unfamiliar input are becoming more and more available. The subject of this study is to construct a text classifier that is able to predict the correct label for tweet message using novel approaches in text classification.

The rest of the report is organized as follows. Section II discusses the structure of the given data. Section III explains the preprocessing steps employed. Section IV elaborates on the fundamental techniques used for text representation. Section V demonstrates the state-of-the-art approaches in text classification applied. Section VI presents the results. The report is concluded in Section VIII.

II. Data Description

The data was obtained from EPFL ML Text Classification 2019 competition at AIcrowd[1]. The train and test datasets consist of 2.5 million and $10,000$ tweets respectively. The original tweets used to contain a smiley face depicting whether the content was positive or negative. Our datasets have these smileys removed, and our goal is to predict them. Fortunately, the classes are well balanced, meaning half of the tweets are positive and the other half are negative.

One problem with the data is that each of the tweets is in a raw textual format, meaning it contains a lot of anomalies, such as lexical and syntactic errors, inconsistent usage of punctuation, elongated words, hashtags, emoticons, etc. Many of these ”anomalies” are due to the fact that tweets are written in an informal language and are part of the Twitter lingo and their existence is expected. Discarding these ”anomalies” is the wrong approach since they are implicit indicators of the sentiment of the tweet, thus a more analytical approach is preferred.

III. Preprocessing

Although the data is said to be preprocessed and separated by single whitespace, upon careful investigation several inconsistencies were discovered. The approach used for preprocessing the raw tweets was inspired by Stanford NLP's tweet tokenizer[2] used in GloVe.

Below, we describe all the preprocessing steps performed on each tweet.

Web Addresses: They usually start with ”www”, ”http”, or ”https”, where in the first case, the communication protocol was hidden. They do not have a particular semantic meaning, but their presence is desirable and thus we replaced them with the <*url*> token.

Emoticons: We can infer a lot about the tweet's sentiment by their usage. They are portrayals of a writer's moods or facial expressions. However, there exist a wide range of character combinations that produce emoticons. In order to capture their meaning more efficiently, we replace them with several tokens, such as: <*lolface*>, <*sadface*>, and <*naturalface*>, to name a few.

Numbers: We noted that the GloVe word vectors do not contain representations for numbers in the Twitter corpora. Instead they use the token <*number*> as their replacement.

[1]https://www.aicrowd.com/challenges/epfl-ml-text-classification-2019/
[2]https://github.com/stanfordnlp/GloVe

Following their example, we replaced all numbers with that particular token.

Hashtags: As a metadata tag, these are words or compounds preceded by the symbol #. In some cases, they could capture semantic information, like in "#lifeisgood". However, finding a way to efficiently split such a text is a difficult task. Therefore, we replaced the text with the <*hashtag*> token.

Elongated Text: With elongation, a stronger statement is conveyed for a particular concept. The way this is achieved is by replacing the end character in the original word with multiple occurrences of that character, like in "happyyy". The way we handle this is by replacing the elongation with <*elong*> whilst keeping the original word.

Repetitions: Like elongation, they too convey a stronger statement, so their presence is desirable. All the repetitions are replaced with the <*repeat*> token.

Multiple Whitespaces: We replaced all occurrences of multiple whitespaces with single whitespace so that our vocabulary would not contain erroneous entries after splitting.

Additional techniques that were considered are stop words removal and lemmatization, however, they did not positively affect the overall performance of the models.

IV. Text Representation

Computers are unable to understand the concepts of words. In order to process natural language, a mechanism for representing text is required. A standard approach in text representation is using vectors of numerical values known as feature vectors. There are different approaches in obtaining such vectors, and in this section, we elaborate on the most common techniques used in this study.

Bag-of-Words: In this model, a text is represented as a multiset of its words, disregarding grammar and even word order but keeping multiplicity. After transforming the text, the most common type of characteristics, or features calculated from this model is the *term frequency*, namely, the number of times a term appears in the text.

However, term frequencies are not necessarily the best representation for the text. Common words like "the", "a", "to" are almost always the terms with the highest frequency in the text. To address this problem, one of the most popular ways to normalize the term frequencies is to weight a term by the inverse of document frequency.

TF-IDF: Term frequency-inverse document frequency is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. TF-IDF's were generated on word, n-gram, and character level for our study.

All of these approaches are bounded to the number of unique words in the corpus, which in some cases has a high order of magnitude and can vary from one task to another. In order to address this issue, we have to define a vector space with a fixed dimension independent of the number of words.

Word Embeddings: They give us a way to use an efficient, dense representation in which similar words have a similar encoding [1]. Importantly, we do not have to define this encoding by hand. An embedding is a dense vector of real numbers, whose dimension is specified at the beginning of the training process. A higher dimensional embedding can thus capture fine-grained characteristics.

In our study, we experimented with both pre-trained embeddings and we trained ones ourselves.

1) *Pre-trained*: GloVe is a model for distributed word representation [2]. The online repository contains dataset of pre-trained word embeddings on Twitter corpora with vocabulary with a size of 1.2 million. However, the corpus used to train the GloVe embeddings is not the same as the one obtained from our dataset. To be precise, they differ approximately by $30,000$ words. Therefore, when an unknown word is encountered, we simply replace it with "unk", representing an unknown entity.
2) *Trained*: fastText[3] is a library for efficient learning of word representations. As an extension to the Word2Vec model, it treats each word as composed of character n-grams, essentially defining the vector for a word as a sum of these character n-grams. Hence, it is able to generate better word embeddings for rare and out of vocabulary words [3].

Previously discussed embedding methods have a significant fundamental limitation – that is, they are context-independent. That means that each word has one fixed representation no matter the context in which it was used. In general, it often happens that the semantic meaning of the word relies on the context. Recent state-of-the-art techniques use clever architectures that mitigate this limitation. ELMo is a method that is primarily based on two layers of long short-term memory (LSTM) – a recurrent neural network (RNN), which have been shown to perform well on NLP tasks [4]. BERT is a newer method, fundamentally different than ELMo as it is based on the transformer architecture [5], which dispenses the recurrences and convolutions entirely and uses solely attention mechanisms [6].

We initially wanted to investigate the performance of these contextual embedding methods. However, the computation of these embeddings is an infeasible task since it requires a huge amount of resources. In fact, computing the contextual embeddings for the whole training dataset with a padded sequence length of 30 and embedding dimension of 1024 requires about 300 GB of data. Moreover, getting the actual embeddings is a slow process. These problems were mitigated by using an approach that uses the context to some lesser degree than ELMo and BERT – sentence embeddings.

Sentence Embeddings: Sent2Vec[4] is a computationally efficient method for general purpose unsupervised sentence rep-

[3]https://github.com/facebookresearch/fastText

[4]https://github.com/epfml/sent2vec

resentations [7]. As such, it can be interpreted as a natural extension of the word-contexts from C-BoW to a larger sentence context, with the sentence words being specifically optimized towards additive combination over the sentence, by means of the unsupervised objective function. The representations were generated using the default parameters for each of the libraries.

All of these methods were used in conjunction with the models to create a text classifier able to produce fine-grained decision boundary in the feature space.

V. Models and Methods

Having our dataset preprocessed and the appropriate representations generated, what remains is applying some of the well-known text classification algorithms, comparing their performance, and choosing the model that has the best accuracy on the given task.

The classification algorithms, that we experimented with fall into two major categories: classical machine learning and deep learning models. The former includes Logistic Regression, Support Vector Machine, Random Forest and Naïve Bayes. The latter includes RNN, such as long short-term memory (LSTM) [8] and gated recurrent unit (GRU) [9].

LSTM: They are RNN, which have been shown to perform well on NLP tasks. More recently, it has been shown that their usage for tweet sentiment prediction gives high accuracy [10].

Our LSTM architecture is as follows. In the beginning, there is Embedding layer, which represents a matrix of dense vectors with a fixed size. We initialize this layer with our representations. Since the embeddings were already pre-trained, we made the parameters of this layer trainable in order to fine-tune the embeddings to our data. Then there is LSTM layer with 200 units to match the dimensionality of the embeddings. The number of units was chosen so that it matches the dimension of our GloVe embeddings. We tried LSTM layers with 100 to 300 units, but there was no significant change in the performance. LSTM was followed by a few Dense layers, which is a fully connected layer, of lengths between 25 and 150. We noticed that the more layers we had, the fewer iterations the network needed to be trained. Since the model is not too complex, the network does not need regularization, thus the presence of dropout probability for the edges did not make significant improvements. We opted for a very small dropout rate of 1-3%. The choice of the optimizer did not make a noticeable difference. We trained multiple models with RMSprop, Adam [11] and Nadam [12]. The activation function on the hidden layers was the same for every layer and it was either hyperbolic tangent or sigmoid. The activation function for the final prediction was always sigmoid.

GRU: It is a relatively new architecture that is very similar to LSTM. More precisely, it controls the flow of information similar to LSTM, but it lacks a memory unit. It has been shown that LSTM is stronger and consistently outperforms GRU [13], [14]. However, the benefit of GRU is that it is computationally more efficient.

Our GRU architecture is almost indistinguishable from the LSTM architecture. The structure of the hidden layers was identical, except for the LSTM layer which we replaced with GRU. The choice of hyperparameters was similar. One difference that we tried is that before the final prediction, we added one more GRU layer followed by a few Dense layers. We could afford the computation since GRU is computationally more efficient than LSTM.

LSTM + GRU: We know that LSTM and GRU are fundamentally similar. Adding more layers will not significantly change the basics of our previous architectures. However, we wanted to increase the complexity of our model. Thus we increased the number of parameters by adding at least 3 LSTM or GRU layers, which are interconnected by multiple Dense layers. With a slightly higher edge dropout probability and similar other hyperparameter choices to the simpler models, we got better performance.

Ensemble: Neural networks are known for being difficult to configure. In fact, they have many hyperparameters such as the number of hidden layers, the number of nodes per layer, activation functions, dropout rate, recurrent dropout rate, optimizer, etc. Furthermore, they are sensitive to the specifics of the training data and may find a slightly different set of weights each time they are trained, i.e. they may explore different local optima, thus resulting in different predictions.

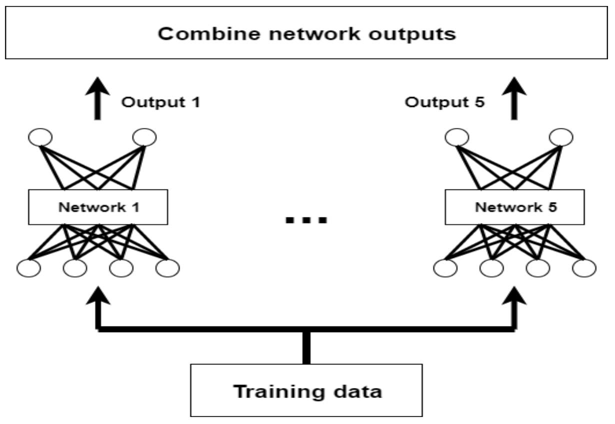

Fig. 1. An ensemble model of neural networks.

A successful approach in reducing the variance of any model is to train multiple models instead of a single model, and combining the predictions of the models. This is called *ensemble learning*. It usually results in superior performance compared to any of the base models [15]. We implemented the majority voting ensemble method. One such configuration is shown on Fig. 1.

With this approach, we have successfully relaxed the extensive search over the hyperparameter space. Now, the only major concern is choosing the best models that will participate in the ensemble. Theoretically, it is possible to find such predictions so that the ensemble predictions of the models will be worse than any of the base models. That is why we were picky with our base models. We chose the top 5 models as the basis for the majority voting ensemble classifier.

TABLE I
CLASSIFICATION ACCURACY OF THE CLASSICAL MACHINE LEARNING MODELS EVALUATED WITH 5-FOLD CROSS-VALIDATION.

Model	BoW	TF-IDF Word	TF-IDF n-gram	TF-IDF Char	Sent2vec	fastText	GloVe
Naïve Bayes	72.67	75.73	75.56	71.27	66.52	63.15	65.50
SVM	68.99	74.36	68.80	75.87	74.80	78.76	78.40
Random Forest	79.72	79.86	76.53	79.84	75.82	79.38	78.60
Logistic Reg.	79.85	80.19	76.88	**81.76**	75.00	76.89	76.52

TABLE II
CLASSIFICATION ACCURACY ON AICROWD OF THE DEEP LEARNING MODELS.

Model	fastText	GloVe
LSTM	87.4	87.6
GRU	86.4	87.7
LSTM + GRU	87.0	88.0
Ensemble	87.2	**88.3**

VI. RESULTS

Table I shows the classification accuracy of multiple word representation techniques trained on different classical machine learning classifiers. The performance was evaluated with 5-fold cross-validation. We only tested some of the predictions on AIcrowd due to submission count limitations. The results were comparable. The hyperparameters were selected with grid search on the parameters' space.

Table II show the classification accuracy on AIcrowd of pre-trained fastText and GloVe word embeddings used with various neural network architectures.

GloVe word embeddings were obtained from the data published by the authors of the method [2]. We used the embeddings trained on tweets and we fine-tuned them for each of the architectures. All other word representations were trained by ourselves on the whole training dataset.

VII. DISCUSSION

Deep learning models outperform classical machine learning approaches by huge margins. It is expected because all of the current state-of-the-art techniques for the NLP tasks are based on neural networks.

The classical machine learning models perform well when compared to the majority classifier on AIcrowd. Naïve Bayes models generally performed the worst. It is because the method relies on a strong assumption that the features are independent. Support Vector Machines are niche models that only work well with sophisticated embedding methods and their performance is heavily dependent on the kernel choice. Random Forest and Logistic Regression models are robust and overall worked the best for any kind of word representations.

Any of the deep learning models were superior to the best of the classical machine learning models. The choice of embeddings made a difference, but the models were more important. LSTM is theoretically strictly stronger than GRU [14]. It was indeed true for fastText embeddings, but our LSTM models were slightly worse than GRU with GloVe. The reason might be the length of our data. In fact, LSTM can theoretically outperform GRU for long sequences of data and can model long-distance relationships. However, tweets are very short strings of textual data. Moreover, GRU is computationally more efficient than LSTM, thus it could be trained on more training samples within a given time.

LSTM + GRU are complex models that combine the benefits of both architectures. These complex models were often robust in the sense that the classification accuracy was consistent and high. They often performed better than the individual models. Each of the complex models explores a specific part of the space and specialized in it, thus the models have high variance. Combining the predictions of the best models reduces the variance, but also uses the unique knowledge of each model. Therefore, the ensemble models have a similar bias to the complex models and reduced variance. That is why they performed the best.

The pre-trained GloVe embeddings were superior to our fastText embeddings. That is because the GloVe model was trained on huge corpora consisting of tweets. However, the difference in performance was negligible when compared to the difference between classical and deep learning models.

VIII. CONCLUSION

In this paper, we presented the preprocessing challenges of handling textual data written in informal language, explained the possible solutions and compared the performance of classical and deep learning models on sentiment classification task and gave research-supported explanations. The deep learning models outperformed the classical machine learning approaches by a huge margin. Main ideas of the ensemble techniques were shown to be true even for our complex neural networks. Combined predictions were noticeably better than predictions of individual classifiers.

We saw that the type of word representations is crucial for performance. The representations we used are context-independent, which have fundamental limitation in the sense that every word has one embedding no matter the context in which it is used. With the recent improvements in word representations – contextual word embeddings, e.g., ELMo and BERT, we believe that our deep learning architectures could reach significantly higher results if we used them. As discussed, we could not implement it because of resource limitations, but it is a compelling idea worth further research.

References

[1] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[2] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[3] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[4] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Advances in Neural Information Processing Systems*, 2017, pp. 6000–6010.

[7] M. Pagliardini, P. Gupta, and M. Jaggi, "Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features," in *NAACL 2018 - Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[9] K. Cho, B. van Merrienboer, Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation." in *EMNLP*, A. Moschitti, B. Pang, and W. Daelemans, Eds. ACL, 2014, pp. 1724–1734.

[10] X. Wang, Y. Liu, S. Chengjie, B. Wang, and X. Wang, "Predicting polarities of tweets by composing word embeddings with long short-term memory," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, 2015, pp. 1343–1353.

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *3rd International Conference for Learning Representations*, 2014.

[12] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[13] D. Britz, A. Goldie, M.-T. Luong, and Q. Le, "Massive exploration of neural machine translation architectures," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 1442–1451.

[14] G. Weiss, Y. Goldberg, and E. Yahav, "On the practical computational power of finite precision RNNs for language recognition," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 740–745.

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.